A Characterization of the Semantics of Logic Programs with Aggregates

Yuanlin Zhang, Maede Rayatidamavandi
Department of Computer Science, Texas Tech University
{y.zhang, maede.rayatidamavandi}@ttu.edu

Abstract

The aggregates have greatly extended the representation power, in both theory and practice, of Answer Set Programming. Significant understanding of programs with aggregates has been gained in the last decade. However, there is still a substantial difficulty in understanding the semantics due to the nonmonotonic behavior of aggregates, which is demonstrated by several distinct semantics for aggregates in the existing work. In this paper, we aim to understand these distinct semantics in a more uniform way. Particularly, by satisfiability, rationality and consistency principles, we are able to give a uniform and simple characterizations of the three major distinct types of answer set semantics.

1 Introduction

By *aggregates* we mean (possibly partial) functions defined on sets of objects of a domain. They have been proven useful in both Database and Logic Programming community. For example, consider the representation of the policy that a class needs a TA (Teaching Assistant) if it has more than 30 students. Assume we have the facts of the form $takes(S, C)$ which denotes that student S takes course C. We also use $needsTA(C)$ to denote that course C needs a TA. Then we can represent the TA policy, using aggregates, as follows:

$$needsTA(C) \text{ :- } |\{S : takes(S, C)\}| > 30$$

where we use the standard math notations of sets, their cardinality and arithmetic relations. Here the cardinality function is an example of an aggregate function and $|\{S : takes(S, C)\}| > 30$ is an aggregate atom. (In the rest of the paper, we use aggregate and aggregate atom interchangeably when there is no ambiguity.) The program is simple and has a clear intuitive meaning.

It is not a surprise that aggregates have enhanced the representation power of Answer Set Programming (ASP) [Gelfond and Lifschitz, 1988], a branch of Logic Programming, that has gained popularity in the knowledge representation community recently. Assuming the prominent feature of ASP is to give simple yet precise semantics of the default negation (also called negation as failure), the default negation can be taken as an aggregate with a very special property (See Section 2). However, when expanding ASP with aggregates, defining the semantics of the new language becomes challenging because the technique used for defining the semantics of default negation is not immediately applicable to general aggregates.

Significant progress has been made to understand the semantics of programs with aggregates in the last decade. Simple and precise semantics have been discovered. However, a challenge is that these semantics do not always agree with each other even on seemingly simple programs. For example, consider the program $p(a) \text{ :- } |\{X : p(X)\}| \geq 0$. According to the semantics by Son et al [Son *et al.*, 2007], called *SPT* semantics here, and that by Faber et al [Faber *et al.*, 2011], called *FPL* semantics here, the program specifies one belief set $\{p(a)\}$ while there is no belief set for this program according to Gelfond and Zhang's semantics [Gelfond and Zhang, 2014], called $Alog$ semantics. Now, consider another program Π_1

$$\begin{aligned}&p(a) \text{ :- } p(b).\\ &p(b) \text{ :- } p(a).\\ &p(a) \text{ :- } |\{X : p(X)\}| \neq 1.\end{aligned}$$

No belief set exists for this program by *SPT* and $Alog$ semantics while *FPL* semantics offers a belief set of $\{p(a), p(b)\}$.

In this paper, we aim to understand the difference among these semantics by some common principles. For simplicity, *we restrict the head of a rule to be a regular atom.* Recall that, in [Gelfond and Kahl, 2014], the semantics of an ASP program is taken as the specification of belief sets (formally called answer sets) of a rational agent equipped with the program. A belief set is a set of atoms which are "justified" to the agent in terms of the program. A "justified" atom is called a belief here. There are three principles behind the justification: *satisfiability principle* which requires every rule of the program to be satisfied by a belief set; *consistency principle* which says that no atom is believed and not believed at the same time; and the *rationality principle* which says that no atom is a belief unless it is forced by the satisfiability principle. There is some small variation between principles here and the original ones in [Gelfond and Kahl, 2014]. These principles are originally employed to derive informal semantics of ASP programs while they are used here to help obtain the definition of formal semantics of ASP with aggregates.

Consider a logic program $p(b) \text{ :- } p(a)$. The satisfiability principle allows for one empty belief set and one belief set of $\{p(a), p(b)\}$. However, the rationality principle will allow only the empty belief set because $p(a), p(b)$ are not *forced* by the satisfiability principle. Now consider another program

$$p(a).$$
$$p(b) :\!-\, p(a).$$

By satisfiability principle, $p(a)$ is forced to be a belief (by the first rule). As a result, the second rule (which is read as if one believes $p(a)$, one has to believe $p(b)$) forces $p(b)$ to be a belief because of the satisfiability principle. These three principles give the classical semantics of ASP without default negation.

Now consider programs with default negation which is denoted by not. The construct $not\ l$, where l is a regular atom, is read as *l is not a belief.* Consider program Π_2:

$$p(a) :\!-\, not\ p(b).$$
$$p(b) :\!-\, not\ p(a).$$

It is now less straightforward to directly obtain a belief set from a program. However, once a set S of atoms is given, it is easy to apply the satisfiability and rationality principle to check if S is a belief set (containing all beliefs), with the understanding that *any atom outside the set is not a belief.* In contrast, we cannot take any atom in the set as a belief because each one needs to be "justified" by the principles. Each atom outside the given set is called a **non-belief**. Consider atom set $\{p(a)\}$. We know that $p(b)$ is not a belief, and by the satisfiability principle, the first rule forces us to believe $p(a)$. Since $p(a)$ is a belief, the second rule is satisfied (vacuously). So, $\{p(a)\}$ is a belief set. Empty set is not a belief set because $p(a)$ and $p(b)$ are non-beliefs and thus the satisfiability of the two rules requires $p(a)$ and $p(b)$ to be beliefs, violating the consistency principle. Consider $\{p(a), p(b)\}$. By this atom set, we do not have that "$p(a)$ is a non-belief" or "$p(b)$ is a non-belief". So, the satisfiability of the rules forces us nothing and thus neither $p(a)$ nor $p(b)$ is a belief. Hence, $\{p(a), p(b)\}$ is not a belief set. In fact, once an atom set is given, the three principles can be used to check if it is a belief set, i.e., they can lead to the classical ASP semantics [Gelfond and Lifschitz, 1988].

Now consider program Π_3 as an example of a program with aggregates:

$$p(a) :\!-\, |\{X : p(X)\}| \geq 0.$$

We understood "*not l*" as "l is not a belief" before, but it does not seem to be immediately extensible to the understanding of an aggregate atom. The main challenge is that the principles do not specify directly when an aggregate atom is believed/forced. In the rest of the paper, we will develop various understanding of when we can say an aggregate atom is believed (or forced). We first introduce the new concepts used in this work, and then give several ways of defining the semantics, namely straightforward approach, monotonicity approach, convexity approach and equivalence based approach. After that we give the hierarchy result on different semantics and the relation between the newly defined semantics and the existing ones. Then we give an alternative characterization of the semantics. We finally discuss the related work and conclude this paper.

2 Preliminaries

We employ the formalism for aggregates proposed by Marek and Truszczynski [Marek and Truszczynski, 2004] because it makes the presentation of our ideas and techniques more intuitive, simple and explicit. To simplify the discussion, we consider only finite programs.

Assume there is a finite set U which is called a **signature**. Every element of U is called a **regular atom**. A **constraint atom** agg, also called **c-atom**, is of the form (D, C), where $D \subseteq U$ and $C \subseteq 2^D$; D is called the **domain** of agg and it is denoted by $dom(agg)$; C is called the *constraint* of agg and we use $con(agg)$ to denote C; and every element of C is called an **allowed set**. Marek and Truszczynski argued that aggregate atoms can be represented as c-atoms. For example, given $U = \{p(a), p(b), q(a)\}$, $|\{X : p(X)\}| \geq 0$ can be represented as a c-atom $(\{p(a), p(b)\}, 2^{\{p(a), p(b)\}})$ where every possible subset of the domain of the c-atom is an allowed set. Intuitively, the constraint part of the c-atom lists all possibilities that make the aggregate atom true. A c-atom is called **elementary** if it is either of the form $(\{a\}, \{\{a\}\})$, or $(\{a\}, \{\{\}\})$. A c-atom of the form $(\{a\}, \{\{a\}\})$ is called a **positive** elementary c-atom and will be simply written as a. A c-atom of the form $(\{a\}, \{\{\}\})$ is called a **negative** elementary c-atom and it will be written as $not\ a$. They are also called **literals**.

A **normal logic program with c-atoms** (or **program** for simplicity), is a finite set of rules of the form:

$$a :\!-\, c_1, \ldots, c_n.$$

where a is a regular atom and each c_i $(i \in \{1, \ldots, n\})$ is a c-atom. For a rule r of the form above, a is called its **head**, denoted by $head(r)$, and $\{c_1, \ldots, c_n\}$ is the **body** of the rule, denoted by $body(r)$. A program is **elementary** if each c_i $(1 \leq i \leq n)$ is an elementary c-atom. An **ASP program** is an elementary program. Given a program P, we use $U(P)$ to denote the set of all regular atoms that occur in P.

A set S of regular atoms **satisfies** a c-atom agg, denoted by $S \models agg$, if $S \cap dom(agg) \in con(agg)$. S **satisfies** a set of c-atoms A if for any $agg \in A$, $S \models agg$. S **satisfies** a rule r, denoted by $S \models r$, if whenever $S \models body(r)$, $S \models head(r)$. We say that S is a **model** of a program P if for any $r \in P$, $S \models r$. A model S of P is **minimal** if no proper subset of S is a model of P.

The **reduct** of an ASP program P wrt a set S of regular atoms, denoted by $R_G(P, S)$, is the program $\{r : r \in P;$ for all $(not\ l) \in body(r), l \notin S;$ for all $l \in body(r), l \in S\}$. S is an **answer set** of an ASP program P if it is a minimal model of $R_G(P, S)$. This is the classical definition of ASP program [Gelfond and Lifschitz, 1988].

We next review the notions in the existing work on the semantics of aggregates in the context of programs with c-atoms.

The ***FPL*** **reduct** of a program P wrt a set S of regular atoms, denoted by $R_F(P, S)$, is the program $\{r : r \in P, S \models body(r)\}$. S is an ***FPL*** **answer set** of P if S is a minimal model of $R_F(P, S)$ [Faber *et al.*, 2011].

Given sets A and S of regular atoms, the set A **conditionally satisfies** a c-atom agg wrt S, denoted by $A \models_S agg$ if

$A \models agg$ and for every I such that $A \cap dom(agg) \subseteq I$ and $I \subseteq S \cap dom(agg)$, we have $I \in con(agg)$. S is an ***SPT* answer set** of P [Son *et al.*, 2007] if $S = T_P^\infty(\emptyset, S)$, where

$$T_P(R, S) = \{a : \exists r \in P, head(r) = a, R \models_S body(r)\}$$

and $T_P^0(\emptyset, S) = \emptyset$ and for any $i \geq 0$,

$$T_P^{i+1}(\emptyset, S) = T_P(T_P^i(\emptyset, S), S).$$

The *Alog* **reduct** of a program P wrt a set S of regular atoms, denoted by $R_A(P, S)$, is the program obtained from P by 1) removing any rule whose body has a c-atom not satisfied by S, and 2) replacing any c-atom, agg, with the set $S \cap dom(agg)$. S is an *Alog* **answer set** of P if it is a minimal model of $R_A(P, S)$. [Gelfond and Zhang, 2014]

3 Semantics of Normal Logic Programs with C-atoms

We first show that the three principles used to define the semantics of Π_2 in the introduction section can be employed to define answer sets of ASP programs.

Given disjoint sets S (intuitively a set of beliefs) and N (intuitively a set of non-beliefs – same in the rest of the paper) of regular atoms, a set B of literals is **forced** by S wrt N if for any positive literal l of B, $l \in S$ and for any negative literal $not\ l \in B$, $l \in N$. Given a sequence $r_1, \ldots, r_n$ of rules of a program P, for any $i \in \{1, \ldots, n\}$, we define $H_i = \{head(r_1), \ldots, head(r_i)\}$, and define $H_0 = \{\}$. The sequence is a **derivation** of P wrt N if for any rule $r_i (i \in \{1, \ldots, n\})$ in the sequence, $body(r_i)$ is forced by H_{i-1} wrt N and for any rule of $P - \{r_1, \ldots, r_n\}$, its body is not forced by H_n wrt N. A set S is a **straightforward answer set** of an ASP program P if there exists a derivation $r_1, \ldots, r_n$ of P wrt $U(P) - S$ such that $H_n = S$. (Note, $U(P) - S$ is the set of all non-beliefs in terms of S.)
Consider the program P below and a set $S = \{a, b, d\}$

$a.$
b :- $not\ c.$
d :- $a, b.$

Let $N = U(P) - S = \{c\}$, i.e., c is a non-belief and we have no beliefs so far ($H_0 = \{\}$). The body of the first rule, denoted by r_1, is forced vacuously. Now $H_1 = \{a\}$. The body of the second rule is forced by H_1 wrt N. Lets denote the second rule by r_2. Now $H_2 = \{a, b\}$. We can verify that the body of the third rule, denoted by r_3, is forced by H_2 wrt N. Then there is no more rules whose body is forced. This sequence r_1, r_2, r_3, is a derivation of the program wrt N. It is easy to verify that S is a straightforward answer set of P because r_1, r_2, r_3 is a derivation of P wrt N and $H_3 = S$.

We can prove the following result on the relation between the new definition of answer sets and the classical one for ASP programs.

Proposition 1 *A set S is a straightforward answer set of an ASP program P iff it is an answer set of P.*

The full proof of all propositions in this paper can be found in [Rayatidamavandi and Zhang, 2016].

In the following subsections, we will extend the definition of answer sets to non-elementary programs. A key question is when we are forced to believe a c-atom, given the beliefs so far and non-beliefs? To see the challenge, consider a c-atom $agg_{e_1} = (\{p(a), p(b)\}, \{\{p(a)\}\})$. Given a belief set $S = \{p(a)\}$ and non-belief set $N = \{\}$, although S satisfies the c-atom, we are not sure if it is still satisfied after we have more beliefs. The case for general c-atoms is not as simple as the elementary c-atoms.

Several types of answers to this question will be discussed in the following subsections.

3.1 Semantics Using only Satisfiability of Aggregates

Let us consider the c-atom $agg_{e1} = (\{p(a), p(b)\}, \{\{p(a)\}\})$ again. We note that given the belief set $S = \{p(a)\}$ and the non-belief set $N = \{p(b)\}$, the c-atom is forced (i.e, always satisfied by S wrt N no matter what new beliefs are produced) if we follow the consistency principle. Based on this observation, a natural generalization of the notion of *forced* from elementary c-atoms to c-atoms is as follows. Given two disjoint sets S and N of regular atoms, a c-atom agg is **forced** by S wrt N if $S \models agg$ and $(dom(agg) - S) \subseteq N$. A set of c-atoms is **forced** by S wrt N if every c-atom of the set is forced by S wrt N. A sequence $r_1, \ldots, r_n$ of rules of a program P is a **derivation** of P wrt N if for any rule $r_i (i \in \{1, \ldots, n\})$ in the sequence, $body(r_i)$ is forced by H_{i-1} wrt N and for any rule of $P - \{r_1, \ldots, r_n\}$, its body is not forced by H_n wrt N. A set S is a **straightforward answer set** of a program P if there exists a derivation $r_1, \ldots, r_n$ of P wrt $U(P) - S$ such that $H_n = S$. We have the following result.

Proposition 2 *A set S is a straightforward answer set of a program P iff it is an Alog answer set of P.*

3.2 Semantics Using Properties of Aggregates

When to answer whether we have to believe a c-atom we may infer that from the properties of the c-atom. Consider program Π_3', the c-atom version of program Π_3 (in the introduction section):

$$p(a) : -(\{p(a)\}, \{\{\}, \{p(a)\}\}).$$

Let the c-atom in the program above be denoted by agg. Assume we try to check if $\{p(a)\}$ is a belief set. Clearly there is no non-beliefs and thus non-belief set N is empty. By the definition of *forced*, agg is not forced intuitively because we do not know whether $p(a)$ is a belief. An alternative way is to make use of the property of the c-atom to decide whether we have to believe a c-atom. For agg, no matter we believe $p(a)$ or not, one of the allowed sets $\{\}$ or $\{p(a)\}$ will make us believe the c-atom. Note we do not have a particular preference of different intuitions in defining answer sets in this paper, and we aim only to provide techniques to "formalize" these intuitions.

Let S and N be sets of ground atoms. A c-atom agg is **M-forced** by S wrt N if 1) it is forced by S wrt N, or 2) $S \models agg$ and $\forall Y$ such that $S \cap dom(agg) \subseteq Y \subseteq dom(agg)$, $Y \models agg$. Intuitively, the second part means that no matter what beliefs we may add to S, agg will always be satisfied. M here denotes the "monotonicity" property. A c-atom agg is **C-forced** by S wrt N if $S \models agg$, and $\forall Y$ such that $(S \cap$

$dom(agg) \subseteq Y \subseteq (dom(agg)) - N, Y \models agg$. Intuitively, C-forced definition makes sure that the c-atom is satisfied by S together with any subset of other beliefs except those in N. C here denotes this "convexity" property. As examples, c-atom $(\{p(a)\}, \{\{\}, \{p(a)\}\})$ is M-forced by $S = \{\}$ wrt $N = \{p(b)\}$; c-atom $(\{p(a), p(b)\}, \{\{\}, \{p(a)\}\})$ is not M-forced by S wrt N but C-forced by S wrt N.

A set A of c-atoms is **M-forced** (and **C-forced** respectively) by S wrt N, if any c-atom of A is M-forced (and C-forced respectively) by S wrt N. A sequence $r_1, \ldots, r_n$ of rules of a program P is an **M-derivation** (and **C-derivation** respectively) of P wrt N if for any rule $r_i (i \in \{1, \ldots, n\})$ in the sequence, $body(r_i)$ is M-forced (and C-forced respectively) by H_{i-1} wrt N and for any rule of $P - \{r_1, \ldots, r_n\}$, its body is not M-forced (and not C-forced respectively) by H_n wrt N. A set S is an **M-answer set** (and **C-answer set** respectively) of a program P if there exists an M-derivation (C-derivation respectively) $r_1, \ldots, r_n$ of P wrt $U(P) - S$ such that $H_n = S$. Since the mechanism underlying these definitions is the same as before, we do not give examples. We have the following result related to the existing work.

Proposition 3 *S is a C-answer set of a program P iff it is an* SPT *answer set of P.*

3.3 Semantics Using Properties of Programs

When deciding whether a c-atom has to be believed under a given set of belief, we can further make use of the knowledge (i.e., other rules) in the given program. Consider the example Π_4 (a simple variation of example 5 [Son and Pontelli, 2007]) where the intuitive meaning of the only c-atom is $|\{X : p(X)\}| \neq 1$:

$p(a)$:–$p(b)$.
$p(b)$:–$p(a)$.
$p(a)$:–$(\{p(a), p(b)\}, \{\{\}, \{p(a), p(b)\}\})$.

The first two rules of the program indicate that $p(a)$ and $p(b)$ belong to an "equivalence set", meaning we either believe none of them or believe both of them. Hence, assuming the program has an answer set, the c-atom in the body of the third rule has to be believed no matter what our beliefs are. To check if $\{p(a), p(b)\}$ is a belief set, starting from empty belief set, we have to believe $p(a)$ by the third rule and then $p(b)$ by the second one. However, in this case, the previous mechanism to define answer sets does not seem to be sufficient to capture this intuition. We need additional concepts including *order preserving* and *equivalence*.

The **dependency graph** of a program P, denoted by G_P, is a directed graph (V, E) where $V = U(P)$ and $(u, v) \in E$ iff there is a rule $r \in P$ and $agg \in body(r)$ such that $u = head(r)$ and $v \in dom(agg)$. Given a directed graph G, a vertex i is **before** a vertex j if there is a non-empty path from j to i in the graph. Two vertices i and j are a **tie** if i is before j and j is before i. Vertex i is **strictly before** vertex j if i is before j and i and j are not a tie. The **tie set** of a regular atom a wrt program P, denoted by $tie(a, P)$, is $\{b :$ b is a vertex of the graph G_P and a and b are a tie in $G_P\}$.

Given a program P and a sequence of rules $r_1, \ldots, r_n$, of P, the sequence is **order preserving** if for any $head(r_i)$ and $head(r_j)$ $(i, j \in \{1, \ldots, n\})$ such that $head(r_i)$ is strictly before $head(r_j)$ in the dependency graph of $\{r_1, \ldots, r_n\}$, $i < j$. More discussion on order preserving can be found in the next subsection.

Given a sequence of rules $r_1, \ldots, r_n$, and a regular atom $head(r_i)$ $(i \in \{1, \ldots, n\})$ such that $T = tie(head(r_i), \{r_1, \ldots, r_n\})$ is not empty, the **last tie index** of $head(r_i)$ denoted by $lastT(head(r_i))$ is the maximal j such that $j \leq n$ and $head(r_j) \in T$. The **first tie index** of $head(r_i)$, denoted by $firstT(head(r_i))$, is the minimal j such that $j \leq n$ and $head(r_j) \in T$. Note that for an atom a, if $tie(a) = \emptyset$, the first and last tie indices are not defined.

We next introduce the equivalence notion behind which the intuition is that a set S of regular atoms is an equivalence set of atoms if any non-empty subset of S can be used to get the rest of atoms in S using the rules of the program. Given a program P and a set A of regular atoms, a set S of regular atoms is an **equivalence set** wrt P and A if for any $T \subset S$ and $T \neq \emptyset$, there exists a rule r of P such that $A \cup T \models body(r)$ and $head(r) \in S - T$. Intuitively A in the definition above is the set of beliefs so far. Consider the following example:

b:–a.
c:–$(\{a, b\}, \{\{a, b\}\})$.
a:–$(\{a, b, c\}, \{\{a, b, c\}\})$.
a:–$(\{a, b, c\}, \{\{b\}, \{a, b, c\}\})$.
a:–$(\{a, b, c\}, \{\{c\}, \{b, c\}, \{a, b, c\}\})$.

We can verify that $\{a, b, c\}$ is an equivalence set wrt $\{\}$. Any non-empty proper subset of $\{a, b, c\}$ can be used to get the rest of the atoms in the equivalence set. For example given $\{a, b\}$, the second rule can be used to derive c, or given $\{b, c\}$ the last rule can be used to derive a.

A sequence $r_1, \ldots, r_n$ of rules of a program P is an **E-derivation** of P if 1) for any $i \in \{1, \ldots, n\}$, $H_{i-1} \models body(r_i)$, 2) the sequence is order preserving, 3) for any atom $head(r_i)(i \in \{1, \ldots, n\})$, $tie(head(r_i), \{r_1, \ldots, r_n\})$ is an equivalence set wrt $\{r_1, \ldots, r_n\}$ and H_{t-1}, where $t = firstT(head(r_i))$, 4) for any rule r_i $(i \in \{1, \ldots, n\})$, $H_n \models body(r_i)$, and 5) there is no rule in $P - \{r_1, \ldots, r_n\}$ whose body is satisfied by H_n. (E in the definition above denotes equivalence based.) A set S of regular atoms is an **E-answer set** of a program P if there exists an E-derivation $r_1, \ldots, r_n$ of P such that $H_n = S$.

In E-derivation, conditions 1 and 5 are from satisfiability principle, implying that the head of a rule whose body is satisfied must be believed. Conditions 2 and 4 come from rationality principle (without this, unnecessary beliefs will be produced). Conditions 3 defines a level of "satisfiability" of a c-atom: a c-atom is "satisfiable" if from the "current" beliefs and non-beliefs, the property of the c-atom and the property of the given program, the c-atom can be "inferred" to be satisfied.

Proposition 4 *S is an E-answer set of a program P iff it is an* FPL *answer set of P.*

To prove the result above, we need a new definition and two properties about *FPL* semantics and E-answer sets. Let P be a program, and S be a *FPL* answer set of P. Also let B be a set of atoms such that $B \subset S$. We say that B can be

expanded to S by P if there is a sequence of rules $r_1 \ldots, r_n$ of *FPL* reduct of P wrt S, (denoted by $R_F(P, S)$) such that

1. $B \cup \{head(r_1), \ldots, head(r_n)\} = S$,
2. $B \cup \{head(r_1), \ldots, head(r_i)\} \models body(r_{i+1})$ for any i where $0 \leq i < n$.

Property 1 *let S be a* FPL *answer set of P. Then for any $B \subset S$, B can be expanded to S.*

Next, we have the minimality property of E-answer sets.

Property 2 *Let P be a program, S be an E-answer set of P and $\{r_1, \ldots, r_n\}$ be an E-derivation such that $H_n = S$. S is a minimal model of $\{r_1, \ldots, r_n\}$.*

Proof sketch of Proposition 4. The full proof is long and we only provide a sketch here.

We first show the necessary condition ($\Longrightarrow$): if S is a E-answer set of P, then S is a *FPL* answer set of P.

By the definition of E-answer set there exists an E-derivation $r_1, \ldots, r_n$ such that $S = H_n$, $S \models body(r_i) (1 \leq i \leq n)$, and there is no rule in $P - \{r_1, \ldots, r_n\}$ whose body is satisfied by S. These together imply that $R_F(P, S) = \{r_1, \ldots, r_n\}$. By Property 2, S is a minimal model of $\{r_1, \ldots, r_n\} = R_F(P, S)$, which implies that S is a minimal model of $R_F(P, S)$. Hence, S is an *FPL* answer set of P by definition. We next prove the sufficient condition

($\Longleftarrow$): assuming S is a *FPL* answer set of P, S is an E-answer set of P.

We first give a procedure to construct a sequence of rules. Let G be the graph induced by S from the dependency graph of $R_F(P, S)$. Let $C_1, \ldots, C_k$ be the strongly connected components of G. Let $O = \{\{a\} : a \in S, \forall i \in \{1, \ldots, k\}, a \notin C_i$. Assume O is of the form $\{C_{k+1}, \ldots, C_m\}$. For any C_i, C_j $(i, j \in \{1, \ldots, m\}, i \neq j)$, C_i is *before* C_j if there is $a \in C_i$ and $b \in C_j$ so that a is before b wrt G. C_i $(i \in \{1, \ldots, m\})$ is *first* if there doesn't exist $C_j (j \in \{1, \ldots, m\})$ such that C_j is before C_i. By definition of C_i's, for any C_i and C_j $(i, j \in \{1, \ldots, m\}$ and $i \neq j)$, there are only three cases: C_i is before C_j, C_j is before C_i or neither C_i is before C_j nor C_j is before C_i. Since G is a directed graph and finite, there must exists C_i which is first. Construct a sequence of rules as follows:

1. Let $i = 0$. $C := \{C_1, \ldots, C_m\}$.
2. If $C = \{\}$, STOP, otherwise, let M be a first set of C.

2.1 By step 1. there must be $a \in M$ such that there is a rule r, $head(r) = a$, for every c-atom agg of $body(r)$, $\{\}$ is an allowed set of agg. Let r_{i+1} be r. Let i be $i + 1$. Let j be i.

2.2 Let $D = \{a\}$. While $M - D \neq \{\}$, repeat the following: By Property 1, $D \cup H_j$ can be expanded to S through $r'_1, \ldots, r'_l$ ($\{r'_1, \ldots, r'_l\} \subseteq R_F(P, S)$). Let r'_e be the first rule such that $head(r'_e) \in M - D$. Let r_{i+1} be r'_e. Let i be $i + 1$. Add $head(r'_e)$ to D.

2.3 Remove M from C. Goto 2.

Let $r_1, \ldots, r_n$ be the sequence obtained from the process above. By the procedure we can first show that for any $i \in \{0, \ldots, n-1\}$
(1) $H_i \models body(r_{i+1})$.
We can also show that $H_n = S$.

Next, construct a new sequence from $r_1, \ldots, r_n$ by the following procedure:
I. Let $r''_1, \ldots, r''_y$ be the rules in $R_F(P, S) - \{r_1, \ldots, r_n\}$. Let t be the sequence $r_1, \ldots, r_n$. Let $j = 1$,
II. While $j \leq y$, let r_i be the rule in t such that $head(r_i) = head(r''_j)$ and $r_i \in \{r_1, \ldots, r_n\}$. Insert r''_j into the sequence t immediately after r_k where k is a minimal number such that $k \geq i$ and $H_k \models body(r''_j)$. Let $j = j + 1$.

Let the new sequence obtained by the above procedure be $r'_1, \ldots, r'_x$. We show $r'_1, \ldots, r'_x$ is an E-derivation of P. First we can show that $H'_x = S$, and for any rule $r \in P - \{r'_1, \ldots, r'_x\}$, $H'_x \not\models body(r)$. Next, by (1) and the construction procedure, we can show for any $r'_i \in \{r'_1, \ldots, r'_x\}$, $H'_{i-1} \models body(r'_i)$. We then show that $r'_1, \ldots, r'_x$ is order preserving by proving that $r_1, \ldots, r_n$ is order preserving. For any $head(r_i)$ and $head(r_j)$ of the sequence such that $i \neq j$ and $head(r_i)$ is strictly before $head(r_j)$ in G, $head(r_i)$ and $head(r_j)$ are not in the same connected component. By the construction procedure of $r_1, \ldots, r_n$, $head(r_i)$ being strictly before $head(r_j)$ implies that rule r_i is produced earlier than r_j, i.e., $i < j$. Hence, the sequence is order preserving. We then can show that by inserting the new rules by the second procedure the order will be still preserved. We next show that any tie set in $r_1, \ldots, r_n$ is an equivalence set. By the first construction procedure, there is a partition of $\{1, \ldots, n\}$ into $\{1, \ldots, k_1\}, \{k_1 + 1, \ldots, k_2\}, \ldots, \{k_{m-1} + 1, \ldots, k_m\}$ such that $C_i = \{r_{k_{i-1}+1}, \ldots, r_{k_i}\}$. For any atom $head(r_i)$, $tie(head(r_i), \{r_1, \ldots, r_n\})$ is equal to some C_j which is an equivalence set wrt $H_{k_{i-1}}$ because of Property 1. We can show that for any $B \subset C_j$, there exists a rule r such that $H_{k_{i-1}} \cup B \models body(r)$ and $head(r) \in C_j - B$. Therefore, $\{r'_1, \ldots, r'_x\}$ is an E-derivation of P, and S is an E-answer set of P. □

3.4 Alternative Characterization

We have the following hierarchy on the different types of answer set semantics.

Proposition 5 *Given a program P, a straightforward answer set of P is an M-answer set of P, and an M-answer set of P is a C-answer set of P which is in turn an E-answer set of P.*

Intuitively order preserving in deriving beliefs imposes a natural requirement by which we should select first, the more "basic" rules whose heads are used in the body of other rules to derive future beliefs. It becomes more natural if we take each rule as a "definition" of its head. The order preserving happens to ensure the rationality principle when several rules are in consideration. In the following, by using order preserving, we offer another characterization of different types of answer sets.

First we introduce a new type of derivation, based on E-derivation. A sequence $r_1, \ldots, r_n$ of a program P is a **canonical derivation** of P if 1) for any $i \in \{1, \ldots, n\}$, $H_{i-1} \models body(r_i)$, 2) the sequence is order preserving, 3) for any

$i, j \in \{1, \ldots, n\}$, $head(r_i) \neq head(r_j)$, (i.e., all rules have distinct heads), and 4) there is no rule r of $P - \{r_1, \ldots, r_n\}$ such that $H_n \models body(r)$ and $head(r) \notin H_n$ (i.e., the sequence is "maximal"). Now we have the following characterization of the four answer set semantics studied in this paper.

Proposition 6 *Given a program P and a set S of regular atoms,*

- *S is a straightforward answer set of P iff there exists a canonical derivation $r_1, \ldots, r_n$ of P such that $H_n = S$ and for any $i \in \{1, \ldots, n\}$, $tie(head(r_i), \{r_1, \ldots, r_n\}) = \{\}$, i.e., there is no loop in the dependency graph of $\{r_1, \ldots, r_n\}$.*
- *S is a C-answer set (and M-answer set respectively) of P iff there exists a canonical derivation $r_1, \ldots, r_n$ of P such that $H_n = S$ and for any $i \in \{1, \ldots, n\}$, and for any r_j such that $head(r_j) \in tie(head(r_i), \{r_1, \ldots, r_n\})$, $body(r_j)$ is C-forced (and M-forced respectively) by H_{j-1} wrt $U(P) - S$.*
- *S is an E-answer set of P iff there exists a canonical derivation $r_1, \ldots, r_n$ of P such that $H_n = S$, for any $i \in \{1, \ldots, n\}$, $H_n \models body(r_i)$ and $tie(head(r_i), \{r_1, \ldots, r_n\})$ is an equivalence set wrt $\{r_1, \ldots, r_n\}$ and H_{t-1} where $t = firstT(head(r_i))$.*

Note that the consistency principle is reflected by the condition of $H_n = S$. The relationship among different types of answer sets may be clearer and more explicit in the result above.

4 Related Work and Conclusion

The nonmonotonicity nature of aggregate atoms makes the understanding of logic programs with aggregates extremely hard, as demonstrated by the variety of distinct semantics researchers have proposed. On the other hand, excellent findings have been discovered in the last decades to enhance our understanding of logic programs with aggregates from many aspects. It includes early work in Database [Klug, 1982] and in Logic Programming [Kemp and Stuckey, 1991]. More recent work on aggregates has been mainly in the context of Answer Set Programming under different formalisms: logic programs with c-atoms [Marek and Truszczynski, 2004; Son and Pontelli, 2007; Son *et al.*, 2007; Shen *et al.*, 2009; Liu *et al.*, 2010], logic programs with aggregates [Niemela *et al.*, 2002; Pelov, 2004; Faber *et al.*, 2011; Gelfond and Zhang, 2014] first-order formulas with answer set semantics [Ferraris and Lifschitz, 2005; Ferraris, 2011; Lee and Meng, 2009; Truszczynski, 2010; Shen *et al.*, 2014] and abstract dialectical frameworks [Brewka *et al.*, 2013; Strass, 2013; Alviano and Faber, 2015]. Most of the semantics are "equivalent" to one of the three semantics (in the c-atom based formalism): *FPL* answer sets [Faber *et al.*, 2011], *SPT* answer sets [Son *et al.*, 2007], and *Alog* answer sets [Gelfond and Kahl, 2014], which is the reason for us to focus on these three types of semantics in this study.

Our definition of straightforward answer set of ASP programs may seem to be similar to the definition of well-supported models of ASP programs [Fages, 1994]. Although, for us, the extension of the former to programs with aggregates seems natural, it is not clear how the well-supported model can be extended for programs with aggregates.

The most relevant work to characterize the semantics of logic programs with aggregates is the *computation based approach* [Liu *et al.*, 2010] and the *unified approach* [Alviano and Faber, 2015]. Both our work and the computation based approach introduce a sequence of rules to derive the beliefs. It is worth to note that the sequence is different from the classical one step provability operator T_P where all rules of a program should be applied to derive a new set of beliefs while the rule sequence approach allows to apply one rule at a time to derive new belief(s). The difference between our work and the computation based approach lies in that the principles involved in constructing the sequence is very different. For example, computation based approach requires that the body of a rule r should be satisfied by additional beliefs derived after rule r while that is not the case in the E-answer set definition here (as an example, see program Π_4 in Section 3.3). Also, computation based approach covers only *SPT* semantics but not *FPL* or *Alog* semantics (we do not see a trivial way to apply computation based approach to *FPL* or *Alog* semantics). By using the traditional one step provability operator, the unified approach is able to cover the *Alog* and *SPT* semantics but not the *FPL* semantics. The unified approach is based on the fixed point of an operator (over a power set of literals), but in our case, each derivation is a sequence of rules. As a result, the derivation is extensible to characterize *FPL* semantics while it is not clear how likely the fixed point based semantics can be extended for the *FPL* one. There is an "equivalence" between the "conditions" defining the *immediate consequence operator* in [Alviano and Faber, 2015] and the concepts of forced and C-forced.

In summary, we discovered that by the very basic satisfiability principle, rationality principle and consistency principle, we can define when a given set of atoms is an answer set of a program, possibly using the properties of aggregate atoms and the program. Particularly, by the alternative characterization (Proposition 6), to derive if a given a set of atoms is an answer set, we simply start with the empty belief set and then select a rule whose body is satisfied by the beliefs so far and it is the first (i.e., the body of other rules may use the head of this rule) such rule. Rules whose heads are a tie may be allowed in this derivation if certain conditions related to the properties ("monotonicity" or "convexity") of the aggregate atoms or program hold. In this view, the relation among distinct semantics such as *Alog*, *FPL* and *SPT* becomes simple and clear. We hope our characterization may offer some insight on the debate of different answer set semantics.

In the future, we plan to study how we can extend our work to cover programs with rules allowing disjunction in their heads. It is also interesting to see whether the answer set characterization techniques in this paper are applicable to the semantics (e.g., [Truszczynski, 2010]) that are different from the four studied here. Furthermore, we will study how the principles in this paper are related to the vicious circle principle [Gelfond and Zhang, 2014; Feferman, 2002; Poincare, 1906]. We would also like to examine our work in the context of FO(ID) [Denecker and Ternovska, 2008].

5 Acknowledgment

The authors' work was partially supported by NSF grant CNS-1359359.

References

[Alviano and Faber, 2015] Mario Alviano and Wolfgang Faber. Stable model semantics of abstract dialectical frameworks revisited: A logic programming perspective. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence. IJCAI Organization, Buenos Aires, Argentina, To appear*, 2015.

[Brewka *et al.*, 2013] Gerhard Brewka, Stefan Ellmauthaler, Hannes Strass, Johannes Peter Wallner, and Stefan Woltran. Abstract dialectical frameworks revisited. In *Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*, pages 803–809. AAAI Press, 2013.

[Denecker and Ternovska, 2008] Marc Denecker and Eugenia Ternovska. A logic of nonmonotone inductive definitions. *ACM transactions on computational logic (TOCL)*, 9(2):14, 2008.

[Faber *et al.*, 2011] Wolfgang Faber, Gerald Pfeifer, and Nicola Leone. Semantics and complexity of recursive aggregates in answer set programming. *Artificial Intelligence*, 175(1):278–298, 2011.

[Fages, 1994] Francois Fages. Consistency of clark's completion and existence of stable models. *Journal of Methods of logic in computer science*, 1(1):51–60, 1994.

[Feferman, 2002] S. Feferman. Predicativity. http://math.stanford.edu/˜feferman/papers/, 2002.

[Ferraris and Lifschitz, 2005] Paolo Ferraris and Vladimir Lifschitz. Weight constraints as nested expressions. *Theory and Practice of Logic Programming*, 5(1-2):45–74, 2005.

[Ferraris, 2011] Paolo Ferraris. Logic programs with propositional connectives and aggregates. *ACM Trans. Comput. Log.*, 12(4):25, 2011.

[Gelfond and Kahl, 2014] Michael Gelfond and Yulia Kahl. *Knowledge Representation, Reasoning, and the Design of Intelligent Agents*. Cambridge University Press, 2014.

[Gelfond and Lifschitz, 1988] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In *Proceedings of ICLP-88*, pages 1070–1080, 1988.

[Gelfond and Zhang, 2014] Michael Gelfond and Yuanlin Zhang. Vicious circle principle and logic programs with aggregates. *TPLP*, 14(4-5):587–601, 2014.

[Kemp and Stuckey, 1991] David B Kemp and Peter J Stuckey. Semantics of logic programs with aggregates. In *ISLP*, volume 91, pages 387–401. Citeseer, 1991.

[Klug, 1982] Anthony Klug. Equivalence of relational algebra and relational calculus query languages having aggregate functions. *Journal of the ACM (JACM)*, 29(3):699–717, 1982.

[Lee and Meng, 2009] Joohyung Lee and Yunsong Meng. On reductive semantics of aggregates in answer set programming. In *Logic Programming and Nonmonotonic Reasoning*, pages 182–195. Springer, 2009.

[Liu *et al.*, 2010] Lengning Liu, Enrico Pontelli, Tran Cao Son, and Miroslaw Truszczynski. Logic programs with abstract constraint atoms: The role of computations. *Artif. Intell.*, 174(3-4):295–315, 2010.

[Marek and Truszczynski, 2004] Victor W Marek and Miroslaw Truszczynski. Logic programs with abstract constraint atoms. In *AAAI*, volume 4, pages 86–91, 2004.

[Niemela *et al.*, 2002] Ilkka Niemela, Patrik Simons, and Timo Soininen. Extending and implementing the stable model semantics. *Artificial Intelligence*, 138(1–2):181–234, Jun 2002.

[Pelov, 2004] Nikolay Pelov. *Semantics of logic programs with aggregates*. PhD thesis, Katholieke Universiteit Leuven, Leuven, Belgium, April 2004.

[Poincare, 1906] H. Poincare. Les mathematiques et la logique. *Review de metaphysique et de morale*, 14:294–317, 1906.

[Rayatidamavandi and Zhang, 2016] Maede Rayatidamavandi and Yuanlin Zhang. Full proof of propositions. https://goo.gl/M3hE5H, 2016.

[Shen *et al.*, 2009] Yi-Dong Shen, Jia-Huai You, and Li-Yan Yuan. Characterizations of stable model semantics for logic programs with arbitrary constraint atoms. *TPLP*, 9(4):529–564, 2009.

[Shen *et al.*, 2014] Yi-Dong Shen, Kewen Wang, Thomas Eiter, Michael Fink, Christoph Redl, Thomas Krennwallner, and Jun Deng. Flp answer set semantics without circular justifications for general logic programs. *Artificial Intelligence*, 213:1–41, 2014.

[Son and Pontelli, 2007] Tran Cao Son and Enrico Pontelli. A constructive semantic characterization of aggregates in answer set programming. *TPLP*, 7(3):355–375, 2007.

[Son *et al.*, 2007] Tran Cao Son, Enrico Pontelli, and Phan Huy Tu. Answer sets for logic programs with arbitrary abstract constraint atoms. *J. Artif. Intell. Res. (JAIR)*, 29:353–389, 2007.

[Strass, 2013] Hannes Strass. Approximating operators and semantics for abstract dialectical frameworks. *Artificial Intelligence*, 205:39–70, 2013.

[Truszczynski, 2010] Miroslaw Truszczynski. Reducts of propositional theories, satisfiability relations, and generalizations of semantics of logic programs. *Artif. Intell.*, 174(16-17):1285–1306, 2010.